AUSTRALIAN OS9 NEWSLETTER

EDITOR :
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

(07) 345-5141

OCTOBER 1989

ATTR Syntax: Attr filename [permissions] Usage : Examine or change the security permissions of a file Opts: -perm = turn off specified permission perm= turn on specified permission -a = inhibi rms : d - directory file s - nc to owner w - write permit to own r - read permit to public pw - te permit to public BACKUP Syntax ge : Copies all data from one de ead error occurs s = single rites BASIC09 Syntax : Basic0 ge BUILD Syntax: Build filenar s from standard input CHD S nge working directory to specifi > Usage: Change execution directory to specified path CMP Syntax: Cmp filename1 filename2 Usage : File comparison utility COBBLER Syntax: Cobbler devname Usage : Creates OS-9 bootstrap file from current boot CONFIG Syntax n disks COPY Syntax data from one fil E Syntax : Date [t Opts: t = specify ame> Usage : Check directory for wor sters -m = save of unused cluster nly -o = print <devname> }<devn Del [-x] filenam s : -x = delete x: Deldir directo yntax: Dir [e] [x] the file names y x=print executi Usage : Display s converted characters to standard output DSAVE Syntax : Dsave [-opts] [dev] [pathname] Usage : Generates procedure file to copy all files in a directory system Opts : -b make a system disk by using OS9boot if present -b=<path> = make system disk using pat do not process b makdir command o num K ECHO Syn tandard output EI riented text edito s text error messages for given error numbers EX Syntax: ex <modname> Usage: Chain to the given module FORMAT Syntax : Format <devname> Usage : Initializes an OS-9 diskette Opts ; R - Ready L - Logical format only "disk name" 1/2 number of sides 'No of

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group

EDITOR : Gordon Bentzen

HELPERS : Bob Devries and Don Berrie

SUPPORT : Brisbane OS9 Level 2 User Group.

Welcome to the October Edition of the Australian OS9 Newsletter.

One of the problems in presenting the Newsletter each month is trying to present sufficient worthwhile material to maintain your interest, while keeping the broad spectrum of readers happy. It's plain to see that while articles about the kernel, sysgo or init modules, or programmes involving complex 'C' functions might be interesting to some, they would be complete double-dutch to others. With that in mind, one of the things which came to mind recently, was the possibility of developing a "wish list" of things for OS9. This could be hardware, software or simply documentation. Let's see if we can't start the ball rolling.

A. Hardware : Anything to do with Hard Disks. What about a battery backed ramdisk. No-halt floppy controllers. How about a way to tidy up the CoCo Multi-Pak, ROMPak Cartridge problem. An extension keyboard. Battery backed system clock.

B. Software : What about a graphics terminal driver. MIDI support for OSK/68000. A screen blanking utility a la MACINTOSH. A high speed terminal program. An intelligent compiler environment programme. User friendly installation programmes for existing software.

C. Documentation : User friendly documentation for system calls. Decent documentation for the 'C' Compiler libraries. A good explanation of device drivers and descriptors. A Basic09 sourcebook for advanced users. A beginners introduction to the operating system, with special reference to basic system setup and customization.

The above list is by no means exhaustive, but represents at least a start. We would appreciate some feedback on the type of thing that you would like. We, at this end of the Newsletter are not magicians, but we think that at least some of the above list are possible, even from our own limited talents.

As I said before, we need to know what you want. We also would like to invite criticism of the Newsletter. Are we producing the kind of material that you like? Do you want more Basic09? More 'C' programmes. How about some assistance with the Microware Assembler? The linker, and it's associated libraries? Would you be prepared to type in Motorola S-Code listings for already assembled programmes? How would you feel if we published the source code for some of the more popular Public Domain programmes? We need your feedback!

What about users or potential users of OS9/68000? Should we produce more material for those readers? Do we have any readers who are interested in OS9/68000 either currently, or in the future? Do you feel the need for further articles of the tutorial type similar to Don Berrie's series on the Macro Text Editor? How do you feel about source code spread out over several issues? Bob Devries' database in 'C' is an example. Would you prefer to have the whole source in one lump?

This of course illustrates one of our problems. Some of the source codes for more useful applications run to fairly large number of pages. The 'C' Database series again serves as an example. A complete issue could have been devoted to just that programme, however if you had no interest in that particular programme, then that particular issue would be wasted as far as your interests are concerned.

Another problem is that while some of our contributors may be only too willing to share the compiled versions of their creations, they may not want the source distributed. This often means that the only way that those programmes could be made available is through our public domain disks. Let us know how you feel about getting some material that way. Remember however, that method would involve more expense.

This month we again present our usual potpourri of listings and suchlike. We have the very last installment of Bob Devries' 'C' Database. A monumental effort. I would like to take this opportunity to thank Bob for sharing this code with us. We also have another in the tutorial series on the OS9 Macro Text Editor, from Don Berrie.

Once again, I remind you that we need from you some kind of feedback. Whether it's comments, criticisms, source material or questions, it's important to us. Until next edition,

Keep computing, Regards, Gordon.

ooooooooooOOOOOOOOOOoooooooooo

EDIT : (Part 3) Using Macros

One of the least used facilities of the built-in line editor is it's ability to use Macros.

What is a macro anyway? Simply put, a macro is a series of commands that an application can process. Of course, most programmes which have this ability, also have the potential for recursion, multiple-step loops and conditional processing. If this sounds to you like a programming language, then you are correct. That is exactly what it is. Simply put, macros are the ability to process instructions from a device or part of memory other than an interactive keyboard.

The OS9 text editor supports the use of this kind of processing. Macros can be used to replace frequently used series of keystrokes, as well as more complex recursive operations. Macros consist of two parts, the header and the body, and are called by the editor by the use of a full stop in column one, followed by the name of a previously defined macro. For example :

```
.macro_name
```

would call the macro previously defined as "macro_name".

The header of the macro defines it's name, as well as any parameters passed to it. Sound familiar? (Almost the same as a Basic09 process!!) The body consists of any number of ordinary edit commands.

The first part of the macro header is the name. The name consists of any number of consecutive letters and underline characters. Even though macro names can be as long as you like, it would seem to me that short names are best. There seems to be no point in typing a twenty letter macro name, when the macro itself might only contain ten or twelve keystrokes. The aim is, of course, to reduce the operator input. Suitable macro names might be :

```
uplo del_all dl ...
```

One thing that the manual doesn't tell you is that there are a number of built in macros. For example "dir eof zero ... ect", in fact any of the 'dot' commands that we have previously dealt with. So you can't define a macro using one of those names. Unfortunately, the programme doesn't tell you this, but simply reports "Macro Already Defined" when you try to exit from the Edit Macros section of the programme.

The second part of the header is the parameter section. Like other edit commands, macros can accept parameters so that they can work with different strings and with different numbers of items. Macros are unable to use parameters directly, that is from within the macro itself. Instead, Edit passes the parameters on to the commands that make up the macro.

As with a Basic09 procedure, when passing parameters to a macro, it is important that the macro knows how many parameters there are, what order they are in, and what type they are (whether numeric or string). This information needs to be defined in the second part of the macro header, commonly called the macro variable list.

Each variable in the macro variable list represents the value of the macro parameter in its corresponding position. There are two acceptable types of macro variables (parameters). These are numeric and string variables. A variable can be defined using any name (with similar restrictions to those of macro names) preceded by a '#' or

'$'. Variables starting with a '#' character are interpreted as numeric variables, while those starting with a '$' are interpreted as strings. Suitable variable names might be :

#N #LONG #iteration_number #abc ... for numeric variables,

and $A $str $STR_A ... for strings.

So the first line in a macro could be :

DLL #TIMES $search_string,

where DLL is the macro name, and #TIMES and $search_string are numeric and string variables respectively.

The body of the macro would then continue. Any edit commands could be used in the body, as well as special macro commands. The special macro commands are :

.MAC<delim>STR<delim> ... STR means any string, for a previously defined macro, within the confines of a pair of similar delimiters. This command opens an existing macro of that name.

.MAC<delim><delim> .. creates a new macro.

Both of the above commands put the editor in the macro editing mode, and the screen shows M: instead of E: to show that the editor is indeed in that mode.

! - Exclamation point. This character is used to place comments inside a macro definition. Any characters between an exclamation point and the next <CR> are assumed to be a comment, and are not processed. This means that you could use a command at the beginning of the line, and then follow it by a comment, on the same line.

Q <CR> - Quits the edit macro made and returns you to the edit mode. The first line of the macro MUST begin with a name that is not already previously defined, in order to close the definition and return to the Edit mode.

.string - Full Stop followed by a string. This is interpreted as a command to execute another macro. Essentially, this is the mechanism for allowing recursion of macros, however you should be very careful when using recursion inside a macro. It's very easy to get yourself into a loop which has no way out!

There are a number of other commands that, while they may be used within a macro, are more properly used for the loading and saving of your macros for future use. Some of these commands are :

.SAVE<delim>STR1<delim>STR2<delim> - This command saves STR1, which may be a single macro name, or a list of macro names delimited by spaces, into the file with a pathname of STR2. The command tries to create this file, and will report an error if it already exists. A bit stupid if you want my opinion.

.LOAD<delim>STR<delim> - This command is used to load a macro file previously saved by the .SAVE command. If during the load, an attempt is made to load from the file, a macro with the same name as one already defined, then this macro is skipped, and the load proceeds from the next macro name.

.DEL<delim>STR<delim> - This command is used to delete a macro named STR in the macro buffer.

.SEARCH and .CHANGE - These commands perform similar functions to their equivalent Edit commands.

The actual process of writing a macro is reasonably simple, especially when the task that it is to perform is simple. Quite complex macros can be built up to perform tasks that would normally require large numbers of keystrokes. I hope to be able to present some macros that you can use with Edit in future issues of the newsletter.

One of the things that I find infuriating with this editor, is the fact that when you use the 'D' command, the damn thing echoes the deleted line, not the current line. The following is a macro that deletes n lines, lists the lines that are deleted, and then lists the current line. To use it, simply type :

```
.dl n
```

where n is the number of lines to delete. I hope you like it. Perhaps this will spur you onto bigger and brighter things. You never know, you might even want to share your creations with the rest of us. Another helpful exercise to do to aid you in the development of macros is to do Sample Session 5, on Page 7-49 of the OS9 Level Two Users Manual (Page 55 of the ORANGE Level One Manual).

Good Luck with your macros, Don Berrie.

<Listing>

```
dl #N
      ! The above line is then Macro Header
V0    ! Turn verify off
[     ! Beginning of loop
D     ! Delete a line
] #N  ! End of loop and loop counter
V1    ! Turn verify back on
L     ! List Current line
```

<End of Listing>

oooooooooo0000000000oooooooooo

The OS9 'C' Compiler.

A review by Bob Devries.

The C compiler programme I have used to compile my programmes for Colour Computer OS9 has been the one sold by Tandy. Its catalogue number is 26-3038, and it is sourced from MicroWare Systems.

The C compiler comes on 2 35 track, single sided, 630 sector Colour Computer OS9 disks as is normal with all Tandy OS9 products. The package also includes a user manual.

The two disks contain all the necessary programmes and data files to enable you to compile a programme from C source code to an OS9 6809 code object module. On the main disk, which is NOT bootable, there is only a CMDS directory which contains:-

```
CC1     C.PREP  C.PASS1 C.PASS2 C.OPT   C.ASM   C.LINK
COPY    DEL     DIR     ECHO    LIST
```

The second disk contains three directories in the root directory,

```
LIB     DEFS    SOURCES
```

The LIB directory contains the standard library clib.l and the start code cstart.r. Both of these are required for compiling programmes.

The DEFS directory contains a number of header files which contain C definitions and other useful information.

The SOURCES directory contains sample source code files, and another directory called SYS which contains some assembler source code files.

Having told you what's on the disks, I'll now tell you a bit about the programmes themselves. The compiler consists of several programmes which are used together, along with a small programme which interprets the command line arguments, and builds a script file to do the actual work.

The first programme used is the pre-processor. Called C.PREP, this programme looks through the source code

and expands macros, adds the include files, and sets up any definitions required.

The next part is the compiler pass one, called C.PASS1. This programme checks the syntax and structure of the source code.

C.PASS2 comes next, and it does the actual changing of the C source code into assembler source code, setting up reserved memory requirements, setting up the strings for the assembler etc. so that the assembler can correctly assemble the code.

C.OPT is a code optimiser which removes any unnecessary code and comments (if any), changes long branches to short branches where it can, and generally tightens up the code.

C.ASM is really a disguised (early) version of the Relocatable Macro Assembler such as the one found on the 'Development System' disk. This section converts the assembler source code into object code, but leaves spaces for the addresses of all the subroutine calls to the standard library and other external modules such as 'cstart.r'.

The linker, C.LINK, now comes along and joins together all of the sections of code. It places 'cstart.r' at the beginning, then the body of the compiled programme, and then checks through the code and adds any library calls to the end of the module. Finally it outputs an OS9 runnable 6809 code programme module.

All these operations are, however, completely automatic, and the user has merely to call the compiler with the command line:-

```
CC1 myfile.c
```

This will result, if there are no errors, in a file called 'myfile' to be placed in the execution directory, along with the C compiler programmes.

There are a number of options available, such as a '-f=' option which will tell the linker to place the final module in whatever path is typed after the option.

```
e.g. CC1 myfile.c -f=/r0/myfile
```

So you can see that this compiler is easy to use, and, if you know how to write nice tidy C source code, you will end up with programmes that work well and are not unreasonably long. The compiler is suitable for both Level One and Level Two OS9.

When I compare this C compiler with others I have used, I do notice that it is no-where near as fussy about syntax as, say, the Lattice C compiler for the Amiga. With the Microware compiler, for example, a pointer to a structure may be replaced with the name of the first element of that structure, something which is a definite no-no in Lattice C.

There are some other little peculiarities. For example, most C compilers require a cast to be used for the 'sizeof' function when using a structure, like this:-

```
size = sizeof(struct struct_name);
```

The MicroWare compiler, however, really mucks things up if you do it that way, and only allows the structure name, like this:-

```
size = sizeof(struct_name);
```

Once you get to know these little funnies, this compiler works well, and outputs the expected code. When you combine this with a number of enhancements in the form of patches to several of the modules, especially the replacement of the 'CC1' with a (C) programme in the public domain library called 'CC' which makes available several other options not normally available, the MicroWare compiler is really worth having.

The MicroWare compiler is the only compiler really produced for Colour Computer OS9, and is certainly

streets ahead in the matter of price. The next available compiler is the MicroWare Level Two compiler for over $300. At the normal retail price of $179.95, this makes the Colour Computer version look very healthy, especially if you can pick it up for the very much discounted prices I have seen lately.

When you add the extra libraries and header files from both the 'Development System' disk and the public domain library, the C compiler really is worth having. Once you have mastered the C language, an art in itself, you can write all sorts of programmes, both small and large, and have them running in no time at all, and, because the result is machine code, they run fast. Seeing your own programmes running really gives you a boost, and makes it a pleasure to write more.

Regards,
Bob Devries.

oooooooooo0000000000ooooooooo

A Database in C

By Bob Devries

At last we have come to the last installment of my database programme. Whew ! That was some marathon effort. Those of you who went to sleep half way through can wake up now, while I think of something else to write. However, before that, I'll have to give you the last part of the programme, and the directions on how to compile it using the MicroWare C compiler.

First off, type up the last three sections of the code, and check that they are correct. Now, to compile the programme, you'll need to have all of the previous sections in the same directory together. I usually use my ramdisk for this, it makes the compiler run a bit faster, and doesn't mess up a disk with the temporary files the C compiler creates.

Now, make sure your C compiler disk(s) are in their correct drives, and that your execution directory is pointed to the C compiler CMDS directory, and the working directory to the directory that has the database files on it. The command to compile is simply this:-

```
CC1 db.c
```

For those of you who are using the modified version, or the programme 'CC' from the public domain library, you may replace 'CC1' with either 'CC' or 'CC2' or whatever you use. This syntax will place the compiled programme into the CMDS directory of your C compiler disk. If you would prefer not to do this, then use the following command line:-

```
CC1 db.c -f=/R0/database
```

This will place the programme in the ramdisk. You may change 'R0' to any other drive or directory. Remember, you must give the complete path.

Now a quick word about the last parts of the programme. The 'amend' function allows the user to change the contents of the currently selected record. This function then calls the save routine to commit the new record to disk. The 'match' function allows the user to find more records with the same name, after the first 'find'. It will continue searching through the database file to see if there are any more records with the same 'Surname'. The last segment of code, 'db.c' is merely a list of '#include' statements to tell the compiler which files to use when compiling. Don't forget that the file 'ansi.h' must be present also. Well, that's about it for this series, as usual, I'm available to answer questions if there are any, but remember, I will not be able to make any major revisions to the code.

```
amend(recno)
int recno;
/* amend() replaces each field with a new entry except where ENTER is */
/* pressed alone (giving a null string) when it will leave the original */
/* field data intact */
{
```

```
        char ch;
        char tempstr[20];       /* temporary string for inputting new data */
        int tempcode;           /* temporary int var for postcode */
        int replace;            /* flag for signalling replace curr record */

        ch = '\0';

        while (TRUE)            /* do forever */
                {
                cursor(14,5);
                gets(tempstr);  /* get user input */
                if (tempstr[0] != '\0')           /* if not a nullstring */
                        strncpy(mail.surname,tempstr,20); /* keep the string */
                cursor(51,5);                     /* and keep going */
                gets(tempstr);
                if (tempstr[0] != '\0')
                        strncpy(mail.firstname,tempstr,20);
                cursor(13,7);
                gets(tempstr);
                if (tempstr[0] != '\0')
                        strncpy(mail.street,tempstr,20);
                cursor(11,8);
                gets(tempstr);
                if (tempstr[0] != '\0')
                        strncpy(mail.city,tempstr,20);
                cursor(12,9);
                gets(tempstr);
                if (tempstr[0] != '\0')
                        strncpy(mail.state,tempstr,3);
                cursor(15,10);
                tempcode = atoi(gets(tempstr));
                if (tempstr[0] != '\0')
                        mail.postcode = tempcode;
                cursor(10,23);
                eraselin();
                cursor(10,23);
                printf("All correct ? y or n ");        /* print prompt */
                while ((ch != 'N') && (ch != 'Y'))
                        ch = toupper(getch());          /* user must press y or n */
                cursor(10,23);
                eraselin();             /* erase message line */
                if (ch == 'Y')          /* if user typed 'y' then exit loop */
                        break;          /* else loop round again */
        }
        recno = save(recno,replace);    /* save the changed record */
}



match(recno)
int recno;
/* match() searches from the current record for more matches of the surname */
/* given in the first find() it either displays the wanted record or */
/* re-displayes the record it started from. match() returns the new (or old)*/
/* record number */
{
```

```
        int temprec;
        char tempname[21];
        long lof,pos;
        char ch;

        temprec = recno;
        strcpy(tempname,mail.surname);  /* keep current surname field */
        pos = ftell(fp);

        if (pos == lastrec*sizeof(mail))
                {
                cursor(10,23);
                printf("No more records.");
                return(temprec);
                }
        do
                {
                pos = ftell(fp);        /* keep reading records until EOF */
                fread(&mail,sizeof(mail),1,fp);
                if (pos == lastrec*sizeof(mail))
                        {
                        cursor(10,23);
                        printf("No more found.");
                        return(temprec);
                        }
                } while (strcmp(mail.surname,tempname) != 0); /* or the match is found */
        recno = ftell(fp)/sizeof(mail);
        return(recno);                  /* tell the calling routine the record number */
}

/* database include file. */
/* use this for the compiler command line argument */

#include "database.c"
#include "scrnmask.c"
#include "delete.c"
#include "save.c"
#include "help.c"
#include "find.c"
#include "insert.c"
#include "usage.c"
#include "match.c"
#include "amend.c"
#include "scrndata.c"

/* The end */
```

COCO-LINK Magazine: Just a reminder for those interested in a wider section of CoCo systems. CoCo-Link is produced in South Australia and is well worth your support. Contact the Editor, Robbie Dalzell on (08) 386 1647 or write;

CoCo-Link 31 Nedland Cres. PT. NOARLUNGA SOUTH S.A. 5167

The bi-monthly magazine will cost $8.00 joining fee plus $12.00 subscription annually.

WHAT HAPPENS TO THE SYSTEM AFTER MY PROGRAM RUNS?

Recently, we have seen a number of programmes, both Commercial and Public Domain, which do a number of 'funny' things to your system. Some of the ones that come to mind are DMTREE/HDMREE and WINDOW WRITER (Copyright) and MAXIC (Public Domain). The following article is presented after a weekend of (so far not totally successfully) trying to restore my system to its normal state after running Window Writer. I fully realise that when one attempts to write a program which will successfully run in a 128K machine, then you are severely limited as to what you can really do. However, surely it's possible to restore the palette register information to what it was? How about giving me back the keyboard attributes ... so that at least the <BREAK> key works, or <CTRL>-W for screen pausing?

I don't particularly want to pick on Window Writer. I think that it is a fine program, and certainly offers more features that any other Color Computer word processor that I have seen. But, after I run it, it leaves my system in such a mess, that I have no choice but to press the 'BLACK BUTTON' and reset the damn thing. I don't think that is good enough.

Let me start at the beginning. A friend of mine phoned me, and asked me to have a look at his machine, as he thought that there was a problem. After travelling to his place, and scrutinising his computer, I found that something strange had indeed happened to it. Upon closer examination, we decided that his problem occurred only after running his new wordprocessing programme, Window Writer.

So what were the symptoms? Well, the most obvious thing was the screen. Light blue characters on a violet background. Odd! Then there was the fact that the <BREAK> key did not work. He normally runs his screen with the page pause set, but all his text files, when listed, scrolled merrily off the screen. Then there was the memory problem. Suddenly, only very little available RAM. And guess what? 'Mdir e' showed the module directory with heaps of modules that he knew nothing about, and with link counts varying from 0 to many.

He loaned me his original disks, and the manuals, to take home to find out what he had done wrong. The simple answer was, nothing!! That unfortunately is how the programme leaves the system. I cannot accept that a serious commercial programmer could be that sloppy. But there it is. Not only that, but the programme actually needs to merge a couple of modules with the Basic09 interpreter module, RUNB. If, like most people, you have merged the Syscall, Inkey and Gfx2 modules with Runb, the you will not be able to get the programme to run at all.

(When the other modules were added to the ones that I already had merged with Runb, the whole thing was greater than three 8k blocks. This does not leave enough room within a 64k logical space to run the programme.)

I then tried a number of ways to try to get around the problem, but as yet, I have not successfully found a way to rectify it. The alterations to the screen and keyboard attributes can be simply overcome (on a 512K system) by opening a device window and running the programme in that. When the programme terminates, simply close the new window, and all the modified attributes die with it.

The problem of the modules in memory is not so simple. If you are familiar with the programme, you will have noticed that the text formatter (for outputting text to a printer) runs as a separate process. I think that the formatter is chained from the original programme. When that part is finished, it chains back to the original programme!!! (When a programme is chained in OS9, it kills the calling process.) You can see that things start to get complicated.

With regrets and apologies, I returned the originals to my friend, after destroying, of course the copy that I had made. It really made me feel thankful that I use Stylograph! Seriously though, by the time the next issue of the Newsletter is printed, I hope to be able to include a 'system clean-up' Basic09 programme (512K users only, unfortunately) for Window Writer, to do the work that the original programmer failed to do.

Should you want any further information you can contact me on (07) 375-3236. Cheers ... Don Berrie.